Modeling and Optimization for the Electric Grid

John D. Siirola
Sandia National Laboratories
Albuquerque, NM  USA

With special thanks to

David L. Woodruff
University of California, Davis

William E. Hart
Jean-Paul Watson
Zev Freidman
Sandia National Laboratories
University of Wisconsin

Jack Ingalls
Stanford University

EWO Seminar, Carnegie Mellon University
25 October 2012
Integrating renewable energy generation

• The grid is managed by
  – Independent System Operators (ISO)
  – Regional Transmission Organizations (RTO)
  – Balancing Authorities

• Operator must balance load and generation at all times
  – Supply demand at lowest possible cost
  – Little to no storage in the grid
  – Unit-specific production ramp limits, startup and shutdown times
  – Disturbances absorbed by (spinning) reserve requirements

• Key challenges:
  – Load variability / forecast errors
  – Variability in non-dispatchable (renewable) generation
Dispatch must match *net load*

Plot reproduced from NREL “Western Wind and Solar Integration Study”
http://www.nrel.gov/electricity/transmission/western_wind.html
“Loss” of weekly periodicity

Plot reproduced from NREL “Western Wind and Solar Integration Study”
http://www.nrel.gov/electricity/transmission/western_wind.html
Significant impact on “base load” generators

Plot reproduced from NREL “Western Wind and Solar Integration Study”
http://www.nrel.gov/electricity/transmission/western_wind.html
Significant gaps in renewable forecasts

Plot reproduced from NREL “Value of Wind Power Forecasting”
http://www.nrel.gov/electricity/transmission/western_wind.html
A word about the example problem…

• (In the US) Sequential markets (run by ISO/RTO):
  – “Unit commitment” (UC) / “Day-ahead Market” (DAM)
    • MIP run ~10 hours before the start of a day
    • Sets on/off state for all generator units hourly for 24 hours
  – “Reliability Unit Commitment” (RUC)
    • MIP run ~8 hours before the start of the day
    • Commits additional generators to meet spinning reserve and reliability (N-1 robustness) requirements
  – “Economic Dispatch” (ED) / “Security-constrained ED” (SCED)
    • “Real-time” markets: LP run hourly / every 5 minutes
    • Set generation levels, prices to meet realized demand

• Problem scale
  – 100’s – 1000’s of buses; 2-3x lines
The Challenge: MP is dense and subtle

Minimize: \[ \sum_t \sum_g \left( c_g P_{g0,t} + c_g^{SU} v_{gt} + c_g^{SD} w_{gt} \right) \]  

S.t. \[ \theta_{\text{min}} \leq \theta_{nct} \leq \theta_{\text{max}}, \quad \forall \, n, c, t \]  
\[ \sum_k P_{kct} - \sum_k P_{kct} + \sum_{g(n)} P_{g0t} = d_{nt}, \quad \forall \, n, c = 0, \text{transmission contingency states } c, t \]  
\[ \sum_k P_{kct} - \sum_k P_{kct} + \sum_{g(n)} P_{gct} = d_{nt}, \quad \forall \, n, \text{generator contingency states } c, t \]  
\[ P_k^{\text{min}} N_{k c} z_{kt} \leq P_k \leq P_k^{\text{max}} N_{k c} z_{kt}, \quad \forall \, k, c, t \]  
\[ B_k(\theta_{nct} - \theta_{nct}) - P_k + (2 - z_{kt} - N_{1 k} c) M_k \geq 0, \quad \forall \, k, c, t \]  
\[ B_k(\theta_{nct} - \theta_{nct}) - P_k - (2 - z_{kt} - N_{1 k} c) M_k \leq 0, \quad \forall \, k, c, t \]  
\[ P_g^{\text{min}} N_{1 g c} u_{gt} \leq P_{gct} \leq P_g^{\text{max}} N_{1 g c} u_{gt}, \quad \forall \, g, c, t \]  
\[ v_{g,t} - w_{g,t} = u_{g,t} - u_{g,t-1}, \quad \forall \, g, t \]  
\[ \sum_{q=t-UT_g+1}^t v_{g,q} \leq u_{g,t}, \quad \forall \, g, t \in \{UT_g, \ldots, T\} \]  
\[ \sum_{q=t-DT_g+1}^t w_{g,q} \leq 1 - u_{g,t}, \forall \, g, t \in \{DT_g, \ldots, T\} \]  
\[ P_{g0,t} - P_{g0,t-1} \leq R^+_g u_{g,t-1} + R^{SU}_g v_{g,t}, \quad \forall \, g, t \]  
\[ P_{g0,t-1} - P_{g0,t} \leq R^-_g u_{g,t} + R^{SD}_g w_{g,t}, \quad \forall \, g, t \]  
\[ P_{gct} - P_{g0,t} \leq R^+_g, \quad \forall \, g, c, t \]  
\[ P_{g0,t} N_{1 g c} - P_{gct} \leq R^-_g, \quad \forall \, g, c, t \]  
\[ 0 \leq v_{g,t} \leq 1, \quad \forall \, g, t \]  
\[ 0 \leq w_{g,t} \leq 1, \quad \forall \, g, t \]  
\[ u_{g,t} \in \{0,1\}, \quad \forall \, g, t \]
The Challenge: MP is dense and subtle

Minimize:
\[
\sum_t \sum_g \left( c_g P_{gy} + c_g^S U v_{gt} + c_g^D w_{gt} \right)
\]

S.t.
\[
\theta_{\text{min}} \leq \theta_{\text{nct}} \leq \theta_{\text{max}}, \quad \forall n, c, t
\]
\[
\sum_k P_{kct} - \sum_k P_{kct} + \sum_{g(n)} P_{gy0t} = d_{nt}, \quad \forall n, c = 0, \text{transmission contingency states } c, t
\]
\[
\sum_k P_{kct} - \sum_k P_{kct} + \sum_{g(n)} P_{gct} = d_{nt}, \quad \forall n, \text{generator contingency states } c, t
\]
\[
P_{\text{min}}^{\text{nct}} N_{1kc} z_{kt} \leq P_{ky0t} \leq P_{\text{max}}^{\text{nct}} N_{1kc} z_{kt}, \quad \forall k, c, t
\]
\[
B_k (\theta_{\text{nct}} - \theta_{\text{mct}}) - P_{kct} + (2 - z_{kt} - N_{1kc}) M_k \geq 0, \quad \forall k, c, t
\]
\[
B_k (\theta_{\text{nct}} - \theta_{\text{mct}}) - P_{kct} - (2 - z_{kt} - N_{1kc}) M_k \leq 0, \quad \forall k, c, t
\]
\[
v_{g,t} - w_{g,t} = u_{g,t} - u_{g,t-1}, \quad \forall g, t
\]
\[
\sum_{q=t-UT+1} v_{g,q} \leq u_{g,t}, \quad \forall g, t \in \{UT, \ldots, T\}
\]
\[
\sum_{q=t-DT+1} w_{g,q} \leq 1 - u_{g,t}, \forall g, t \in \{DT, \ldots, T\}
\]
\[
P_{g0} - P_{g0,t-1} \leq R^+_{g} u_{g,t-1} + R^S_{g} v_{g,t}, \quad \forall g, t
\]
\[
P_{g0,t-1} - P_{g0,t} \leq R^-_{g} u_{g,t} + R^D_{g} w_{g,t}, \quad \forall g, t
\]
\[
P_{gct} - P_{g0,t} \leq R^+_{g}, \quad \forall g, c, t
\]
\[
P_{g0,t} N_{1gc} - P_{gct} \leq R^-_{g}, \quad \forall g, c, t
\]
\[
0 \leq v_{g,t} \leq 1, \quad \forall g, t
\]
\[
0 \leq w_{g,t} \leq 1, \quad \forall g, t
\]
\[
u_{g,t} \in \{0,1\}, \quad \forall g, t
\]

To a first approximation:
- DCOPF
- Economic dispatch
- Unit commitment
- Transmission switching
- N-1 contingency

(Nonobvious) Inherent structure

Key feature: Layered (nested) model complexity

N-1 Economic Dispatch

contingencies
nominal case

Unit Commitment
This still doesn’t *quite* tell the whole story
Block-oriented modeling

• “Blocks”
  – Collections of model components
    • Var, Param, Set, Constraint, etc.
  – Blocks may be arbitrarily nested

• Why blocks?
  – Support reusable modeling components
  – Express distinctly modeled concepts as distinct objects
  – Manipulate modeled components as distinct entities
  – Explicitly expose model structure (e.g., for decomposition)

• Prior art
  – Ubiquitous in the simulation community
  – Rare in Math Programming environments
    • Notable exceptions: ASCEND, JModelica.org
    • This is more than just suffixes!
Coopr: a CCommon Optimization Python Repository

Decomposition Strategies
- Progressive Hedging
- Generalized Benders
- DIP Interface (coming soon)

Language extensions
- Disjunctive Programming
- Stochastic Programming
- DAE Modeling (coming soon)

PYthon Optimization Modeling Objects

Core Optimization Infrastructure

Pluggable Solver Interfaces

CPLEX
Gurobi
Xpress
GLPK
CBC
PICO
OpenOpt
AMPL Solver Library
Ipopt
KNITRO
Coliny
BONMIN
Pyomo overview

• Formulating optimization models natively within Python
  – Provide a natural syntax to describe mathematical models
  – Formulate large models with a concise syntax
  – Separate modeling and data declarations
  – Enable data import and export in commonly used formats

• Highlights:
  – Clean syntax
  – Python scripts provide a flexible context for exploring the structure of Pyomo models
  – Leverage high-quality third-party Python libraries, e.g., SciPy, NumPy, MatPlotLib

```python
from coopr.pyomo import *

m = ConcreteModel()

m.x1 = Var()

m.x2 = Var(bounds=(-1,1))

m.x3 = Var(bounds=(1,2))

m.obj = Objective(
    sense = minimize,
    expr = m.x1**2 + (m.x2*m.x3)**4 +
           m.x1*m.x3 + m.x2 +
           m.x2*sin(m.x1+m.x3) )

model = m
```
Rethinking RUC: a “Tinkertoy” approach

• Capture connected block structure, e.g., network flow

– Embed physical component models within separate blocks
– Connect blocks using conceptual interfaces:
  • Connectors: groups of named numeric values
    – Constant, Parameter, Variable, Expression
  • “Connect” connectors with simple constraints
**Simple input-output blocks**

```python
def dc_line_rule(line, id):
    line.B         = Param()
    line.Limit     = Param()
    line.Angle_in  = Var()
    line.Angle_out = Var()
    line.Power     = Var( bounds= ( -line.Limit, line.Limit ) )

    line.power_flow = Constraint( expr=

    line.IN  = Connector( initialize=

    line.OUT = Connector( initialize=
        { "Power": line.Power, "Angle": line.Angle_out } )
```

DC Line
Arbitrary inputs: conservation blocks

```python
def dc_bus_rule(bus, id):
    bus.D = Param()
    bus.Angle = Var()
    bus.Power = VarList()

def _power_balance(bus, P):
    return summation(P) == bus.D

bus.BUS = Connector( initialize={ "Angle": bus.Angle })
bus.BUS.add( bus.Power, "Power", aggregate=_power_balance )
```

- The `VarList` provides a unique local variable for every connection.
- The `aggregation` rule is called after expanding the connections.
General power flow model

```python
from power_flow import \
    dc_line_rule as line_rule, \
    dc_bus_rule as bus_rule, \
    dc_generator_rule as generator_rule

model.BUSES = Set()  
model.LINES = Set()  
model.GENERATORS = Set()

model.links = Param( model.LINES, ['IN', 'OUT'] )  
model.bus = Block( model.BUSES, rule=bus_rule )  
model.line = Block( model.LINES, rule=line_rule )  
model.generator = Block( model.GENERATORS, rule=generator_rule )

def _network(model, l):
    yield model.line[l].IN == model.bus[ value(model.links[l, 'IN']) ].BUS  
    yield model.line[l].OUT == model.bus[ value(model.links[l, 'OUT']) ].BUS  
    yield ConstraintList.End

model.network = ConstraintList( model.LINES, rule=_network )

def _generator_placement(model, g):
    return model.generator[g].OUT == model.bus[ value(model.generator[g].bus) ].BUS

model.generator_placement = Constraint( model.GENERATORS, rule=_generator_placement )

Only domain-specific component (Note: we have only shown the line and bus rules and not the generator rule)
```
So, what’s really happening?

1) Construct hierarchical model
   – Generate blocks (Variables + Internal constraints)
   – “Connect” blocks by forming constraints over block connectors

2) An automatic *model transformation* “flattens” the model
   – Replicates connector constraints for each variable in connector
   – Generates aggregating constraints
   – (Eliminates redundant variables)
Leveraging components: AC power flow

```python
from power_flow import 
    ac_line_rule as line_rule, \
    ac_bus_rule as bus_rule, \
    ac_generator_rule as generator_rule

model.BUSES = Set()
model.LINES = Set()
model.GENERATORS = Set()

model.links = Param( model.LINES, ['IN', 'OUT'] )
model.bus = Block( model.BUSES, rule=bus_rule )
model.line = Block( model.LINES, rule=line_rule )
model.generator = Block( model.GENERATORS, rule=generator_rule )

def _network(model, l):
    yield model.line[l].IN == model.bus[ value(model.links[l, 'IN']) ].BUS
    yield model.line[l].OUT == model.bus[ value(model.links[l, 'OUT']) ].BUS
    yield ConstraintList.End

model.network = ConstraintList( model.LINES, rule=_network )

def _generator_placement(model, g):
    return model.generator[g].OUT == model.bus[ value(model.generator[g].bus) ].BUS

model.generator_placement = Constraint( model.GENERATORS, rule=_generator_placement )
```
Manipulating model blocks

- Generalized Disjunctive Programming (GDP)
  - Switching entire blocks on/off through binary variables

- Introduce new Pyomo modeling components:
  - “Disjunct”
    - a new form of model block
  - “Disjunction”
    - a new constraint for enforcing logical XOR over disjunctive sets

\[
\begin{align*}
\min & \quad \sum_k c_k + f(x) \\
\text{s.t.} & \quad g(x) \leq 0 \\
& \quad \bigvee_{i \in D_k} Y_{ik} \\
& \quad h_{ik}(x) \leq 0 \\
& \quad c_k = \ y_{ik} \\
\end{align*}
\]

\[
(Y) = \text{true} \\
Y_{ik} \in \{\text{true, false}\}
\]
Creating a “switchable line”

- Sidebar: we need an “open line” model

```python
def open_dc_line_rule(line, id):
    line.Limit = Param()
    line.Angle_in = Var()
    line.Angle_out = Var()
    line.Power = Var( bounds=( -line.Limit, line.Limit ) )

    line.power_flow = Constraint( expr= line.Power == 0 )

    line.IN = Connector( initialize=

    line.OUT = Connector( initialize=
                          { "Power": line.Power, "Angle": line.Angle_out } )
```
def switchable_dc_line_rule(line):
    line.CLOSED = Disjunct(rule=dc_line_rule)
    line.OPENED = Disjunct(rule=open_dc_line_rule)
    line.switch = Disjunction(expr=[line.CLOSED, line.OPENED])

    line.FROM = Connector()
    line.TO = Connector()

def connections_rule(line, id):
    yield line.FROM == line.CLOSED.FROM
    yield line.FROM == line.OPENED.FROM
    yield line.TO == line.CLOSED.TO
    yield line.TO == line.OPENED.TO
    yield ConstraintList.End

    line.connections = ConstraintList(rule=connections_rule)
Creating a transmission switching model

```python
from power_flow import 
    switchable_dc_line_rule as line_rule, 
    dc_bus_rule as bus_rule, 
    dc_generator_rule as generator_rule

model.BUSES = Set()
model.LINES = Set()
model.GENERATORS = Set()

model.links = Param( model.LINES, ['IN', 'OUT'] )
model.bus = Block( model.BUSES, rule=bus_rule )
model.line = Block( model.LINES, rule=line_rule )
model.generator = Block( model.GENERATORS, rule=generator_rule )

def _network(model, l):
    yield model.line[l].IN == model.bus[ value(model.links[l, 'IN']) ].BUS
    yield model.line[l].OUT == model.bus[ value(model.links[l, 'OUT']) ].BUS
    yield ConstraintList.End

model.network = ConstraintList( model.LINES, rule=_network )

def _generator_placement(model, g):
    return model.generator[g].OUT == model.bus[ value(model.generator[g].bus) ].BUS

model.generator_placement = Constraint( model.GENERATORS, rule=_generator_placement )
```
Solving GDP models

- Automated transformations generate “flat” MI(N)LPs
  - Big-M relaxation
  - Convex hull relaxation
Putting it all together: UC + switching + N-1

Network Model

Switchable Transmission Line

Bus model

Switchable Generator

Start-Up Model

Generation Model

Ramp Limits ($Y_i$)

Transmission Line Power Flow Model

Current Balance (KCL)

$Y_{i}^{SU}$

$Y_{i}^{G}$

$Y_{i}^{SU} | Y_{i}^{G}$

V

V

V

Sandia National Laboratories
## The cost of flexibility: transformation time

<table>
<thead>
<tr>
<th>Instance</th>
<th>Flat</th>
<th>Blockaded</th>
<th>Blocked+GDP (line only)</th>
<th>Blocked+GDP (line + generator)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-bus (24-hour)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instantiation</td>
<td>0.88</td>
<td>4.75</td>
<td>5.56</td>
<td>12.2</td>
</tr>
<tr>
<td>Connector expansion</td>
<td>–</td>
<td>4.71</td>
<td>5.45</td>
<td>6.1</td>
</tr>
<tr>
<td>GDP transformation</td>
<td>–</td>
<td>–</td>
<td>3.11</td>
<td>8.2</td>
</tr>
<tr>
<td>Total</td>
<td>2.14</td>
<td>14.1</td>
<td>19.3</td>
<td>34.3</td>
</tr>
<tr>
<td>RTS-96 (4-hour)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instantiation</td>
<td>39.4</td>
<td>205</td>
<td>231</td>
<td>427</td>
</tr>
<tr>
<td>Connector expansion</td>
<td>–</td>
<td>206</td>
<td>224</td>
<td>184</td>
</tr>
<tr>
<td>GDP transformation</td>
<td>–</td>
<td>–</td>
<td>139</td>
<td>362</td>
</tr>
<tr>
<td>Total</td>
<td>67.6</td>
<td>574</td>
<td>788</td>
<td>1280</td>
</tr>
<tr>
<td>RTS-96 (8-hour)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instantiation</td>
<td>73.8</td>
<td>386</td>
<td>459</td>
<td>908</td>
</tr>
<tr>
<td>Connector expansion</td>
<td>–</td>
<td>389</td>
<td>449</td>
<td>272</td>
</tr>
<tr>
<td>GDP transformation</td>
<td>–</td>
<td>–</td>
<td>292</td>
<td>733</td>
</tr>
<tr>
<td>Total</td>
<td>148</td>
<td>1130</td>
<td>1610</td>
<td>2600</td>
</tr>
<tr>
<td>RTS-96 (12-hour)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instantiation</td>
<td>110</td>
<td>571</td>
<td>687</td>
<td>*</td>
</tr>
<tr>
<td>Connector expansion</td>
<td>–</td>
<td>657</td>
<td>679</td>
<td>*</td>
</tr>
<tr>
<td>GDP transformation</td>
<td>–</td>
<td>–</td>
<td>434</td>
<td>*</td>
</tr>
<tr>
<td>Total</td>
<td>219</td>
<td>1740</td>
<td>2470</td>
<td>*</td>
</tr>
</tbody>
</table>
Expanded constraints: *presolve required*

<table>
<thead>
<tr>
<th>Model</th>
<th>Raw generated model</th>
<th>After CPLEX presolve</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rows</td>
<td>Columns</td>
</tr>
<tr>
<td>Flat</td>
<td>20923</td>
<td>6697</td>
</tr>
<tr>
<td>Blocked</td>
<td>46979</td>
<td>34961</td>
</tr>
<tr>
<td>Blocked+GDP (L)</td>
<td>64833</td>
<td>50943</td>
</tr>
<tr>
<td>Blocked+GDP (L+G)</td>
<td>88689</td>
<td>54142</td>
</tr>
</tbody>
</table>

[5-bus, 24-hour test case]
The key challenge is managing uncertainty

- Historically, absorbed by (spinning) reserves
  - Nominally, 5-10% base demand
  - Approximates the “true” constraint: reliability requirements
  - Absorbing non-dispatchable generation requires significantly higher reserves due to poor forecasts

- Alternative: directly model reliability requirement
  - Robust optimization (e.g., N-1)
  - Stochastic programming + “appropriate” expectation
    - Optimize expectation over a sufficiently large set of scenarios

- Challenges:
  - Multiple stages
  - Integer variables at any stage
  - Enormous scenario trees
Forming stochastic programs

- Exploit “block diagonal” structure
  - Deterministic model, $M$
  - Replicated for each scenario
  - Coupled by nonanticipativity constraints, $N$

\[
M(x_1, y_1) \rightarrow M(x_2, y_2) \rightarrow \ldots \rightarrow M(x_n, y_n) \rightarrow N(y)
\]
What about when the extensive form is too big / hard?

- Progressive Hedging (PH) [Rockafellar & Wets]
  - Solve scenarios independently
  - Iteratively converge nonanticipativity constraints
  - PySP: generic implementation of PH
    - Automatic problem construction
    - Numerous tricks / heuristics for handling integer decisions
    - Parallelization on large clusters
Applying PH to the $N$-1 problem

• CPLEX can solve the EF at the root node (for our test cases)
  – …using heuristics
  – …in 3 days                      [RTS-96 test case, with 217 contingencies]

• Scenario generation is slightly more complex
  – Choice of decomposition axis: contingency or time?
  – Bundles: nominal case + 1 contingency

• Using PH:
  – The good news:
    • Root nodes solve in < 1 minute
    • This parallelizes “trivially”
  – The bad news
    • Individual scenarios enter the B&B tree
    • … with a relatively large gap (>30%)
  – This is the focus of ongoing research; “so stay tuned”
“Blocks” fundamentally change modeling

- Explicit model blocks
  - Component reuse
  - Implicit transformations when generating model instances

- Generalized Disjunctive Programs
  - Explicit transformations to create standard forms
  - (Solver-specific decomposition)

- Block diagonal models
  - Implicit transformation to create standard forms
  - Solver-specific decompositions (e.g., progressive hedging)

- BUT… a parting shot:
  - The real problem is the ACOPF (nonconvex nonlinear)
  - Actually solving that problem is “nontrivial”
Acknowledgements

• Sandia National Laboratories
  – Bill Hart
  – Jean-Paul Watson
  – John Sirola
  – David Hart
  – Tom Brounstein
• University of California, Davis
  – Prof. David L. Woodruff
  – Prof. Roger Wets
• Texas A&M University
  – Prof. Carl D. Laird
  – Daniel Word
  – James Young
  – Gabe Hackebeil
• Carnegie Mellon University
  – Bethany Nicholson
• Texas Tech University
  – Zev Friedman
• Rose Hulman Institute
  – Tim Ekl
• William & Mary
  – Patrick Steele
• North Carolina State
  – Kevin Hunter

Plus our many users, including:
- University of California, Davis
- Texas A&M University
- University of Texas
- Rose-Hulman Institute of Technology
- University of Southern California
- George Mason University
- Iowa State University
- N.C. State University
- University of Washington
- Naval Postgraduate School
- Universidad de Santiago de Chile
- University of Pisa
- Lawrence Livermore National Lab
- Los Alamos National Lab
For more information…

• Project homepage
  – http://software.sandia.gov/coopr

• “The Book”

• Mathematical Programming Computation papers
  – PySP: Modeling and Solving Stochastic Programs in Python (Vol. 4, No. 2, 2012)